# Ejercicio 1

Programa que lee de la entrada estándar una secuencia de nombres y los guarda en una colección de forma que ésta no contenga nombres repetidos. La primera línea de entrada contiene la cantidad de nombres. El resto de las líneas contienen los nombres a razón de uno por línea. El programa finalizará mostrando los nombres guardados en la colección en el mismo orden en que se insertaron.



**package** unidad8\_colecciones;

**import** java.util.Collection;

**import** java.util.LinkedHashSet;

**import** java.util.Scanner;

**public** **class** Ejercicio1 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner in = **new** Scanner(System.***in***);

Collection<String> lista = **new** LinkedHashSet<>(); // Los elementos se encuentran en el orden que se insertan, sin valores repetidos

**int** n = in.nextInt(); // Número de elementos a insertar

**for** (**int** i=0; i<n; i++) {

String nombre = in.next(); // Almacenamos texto por teclado

**if** (!lista.contains(nombre)) // Comprueba si la lista NO contiene el texto (no hace falta si se usa LinkedHashSet<>)

lista.add(nombre); // Lo agregamos

}

System.***out***.println(lista); // Se imprime lista

in.close();

}

}

# Ejercicio 2

Crea un programa que realice las tareas siguientes:

• Almacenar en una lista 100 números aleatorios entre 1 y 100 y mostrarlos todos utilizando uno de los métodos de iteración.

• Almacenar los números de la lista original en otra colección sin que se repita ninguno y mostrarlos todos usando un método de iteración diferente al anterior.

• Almacenar los números de la lista original en otra colección en la que se almacenen ordenados y sin que se repita ninguno y mostrarlos todos usando un método de iteración diferente al anterior.

**package** unidad8\_colecciones;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Random;

**import** java.util.Scanner;

**import** java.util.Set;

**import** java.util.TreeSet;

**public** **class** Ejercicio2 {

**static** List<Integer> *listaNumeros* = **new** ArrayList<Integer>();

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner in = **new** Scanner(System.***in***);

**int** n = in.nextInt();

*generaNumeros*(n);

/\*

// Usando iterador

System.out.println("\nUSANDO ITERADOR\n");

Iterator<Integer> i = listaNumeros.iterator();

while (i.hasNext()) {

System.out.println(i.next());

}

// Usando for mejorado

System.out.println("\nUSANDO FOR MEJORADO\n");

for (int num: listaNumeros) {

System.out.print(num);

}\*/

// Usando forEach

System.***out***.println("\nUSANDO FOREACH");

*listaNumeros*.forEach(num -> System.***out***.print(num + " "));

// Solo registros únicos

Set<Integer> unicos = **new** LinkedHashSet<>(*listaNumeros*);

System.***out***.println("\nLinkedHashSet");

unicos.forEach(num -> System.***out***.print(num + " "));

System.***out***.println("\nHashSet");

Set<Integer> set1 = **new** HashSet<>(*listaNumeros*);

**for** (Integer a: set1)

System.***out***.print(a + " ");

System.***err***.println();

System.***out***.println("\nTreeSet");

Set<Integer> set2 = **new** TreeSet<>(*listaNumeros*);

set2.forEach(a -> System.***out***.print(a + " "));

System.***err***.println();

in.close();

}

**static** **void** generaNumeros(**int** n) {

**for** (**int** i=0; i<n; i++) {

Random r = **new** Random();

**int** aleatorio = r.nextInt(9)+1;// Random entre 0 y 99 + 1 (de 1 a 100).

*listaNumeros*.add(aleatorio);

}

System.***out***.println("Generada lista");

}

}

# Ejercicio 3

Crea un programa que lea de la entrada estándar una línea de texto y a continuación almacene en una colección las palabras que no se repiten y en otra colección las que sí se repiten. El programa finalizará mostrando el contenido de ambas colecciones (hacerlo sin escribir código para iterar).

**package** unidad8\_colecciones;

**import** java.util.ArrayList;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.Set;

**public** **class** Ejercicio3 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Scanner in = **new** Scanner(System.***in***);

String texto = in.nextLine();

in.close();

String [] palabras = texto.split("\\P{L}+");

Set<String> noRepetidos = **new** LinkedHashSet<>();

List<String> repetidos = **new** ArrayList<>();

**for** (String a: palabras){

//System.out.println(a);

noRepetidos.add(a);

repetidos.add(a);

}

System.***out***.println("No repe: " + noRepetidos);

System.***out***.println("Repe: " + repetidos);

}

}

# Ejercicio 4

Definir una clase Palabras para almacenar palabras según las especificaciones siguientes:

* • No se almacenarán palabras repetidas.
* • Las palabras se almacenarán en una estructura de datos que facilite la obtención de la lista de palabras de una determinada longitud en orden alfabético.
* • Constructores: un constructor sin parámetros que cree la estructura vacía y un constructor que declare un parámetro de tipo String, cree la estructura y almacene en ella las palabras que formen parte del parámetro.
  1. • Métodos para: o Añadir una palabra.
  2. o Añadir las palabras que formen parte de una cadena de caracteres.
  3. o Comprobar si una palabra está contenida en la estructura de datos.

Escribir un programa que ponga a prueba la clase Palabras. El programa mostrará en la consola un indicador para que el usuario introduzca los comandos siguientes (cada comando finaliza con la pulsación de la tecla intro):

* • añadir: … Se almacenarán las palabras escritas a continuación de los dos puntos.
* • lista n Se mostrará la lista de palabras de longitud n.
* • borrar Borra todas las palabras almacenadas.
* • borrar: … Sustituye las palabras almacenadas anteriormente por las palabras escritas a continuación de los dos puntos.
* • fin Borra todas las palabras almacenadas.

## Solución Vero con TreeSet

### Clase Palabras

**package** unidad8\_colecciones;

**import** java.util.Collection;

**import** java.util.TreeSet;

**public** **class** Palabras {

// atributos

**private** **static** Collection<String> *lista*;

// constructor

**public** Palabras() {

*lista* = **new** TreeSet<String>();

}

**public** Palabras(String palabra) {

**this**();

setPalabra(palabra);

}

// metodos

**public** **void** setPalabra(String palabra) {

*lista*.add(palabra);

}

**public** **void** setPalabras(String cadena) {

String [] palabras = cadena.split("\\s+");

**for** (String palabra: palabras){

*lista*.add(palabra);

}

}

**public** **boolean** comprueba(String palabra) {

**if** (*lista*.contains(palabra)) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **void** delPalabra(String palabra) {

*lista*.remove(palabra);

}

**public** **void** clearAll() {

*lista*.clear();

}

**public** String longPalabras(**int** n) {

String resultado = "";

**for** (String palabra: *lista*) {

**if**(palabra.length()==n) {

resultado += palabra + " ";

}

}

**return** resultado.trim();

}

@Override

**public** String toString() {

String txt = "";

**for** (String a: Palabras.*lista*) {

txt += a + " ";

}

**return** txt;

}

// pongo a prueba la clase

/\*

public static void main(String[] args) {

Palabras nuevaPalabra = new Palabras("uno");

nuevaPalabra.setPalabra("dos");

nuevaPalabra.setPalabras("tres cuatro");

nuevaPalabra.setPalabra("dos");

nuevaPalabra.delPalabra("uno");

System.out.println(nuevaPalabra.comprueba("tres"));

System.out.println(nuevaPalabra);

nuevaPalabra.clearAll();

System.out.println(nuevaPalabra);

}

\*/

}

### Clase Ejercicio4

**package** unidad8\_colecciones;

**import** java.util.Scanner;

**public** **class** Ejercicio4 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Palabras lista = **new** Palabras(); // constructor clase Palabras

Scanner in = **new** Scanner(System.***in***);

String teclado;

**do** {

System.***out***.println("Escriba comando. >");

teclado = in.nextLine();

String [] comandos = teclado.split("\\s+");

**switch** (comandos[0].toLowerCase()) {

**case** "borrar:":

lista.clearAll();

/\*for (int i=1; i<comandos.length; i++)

lista.delPalabra(comandos[i]);

break;

\*/

**case** "añadir:":

**for** (**int** i=1; i<comandos.length; i++)

lista.setPalabra(comandos[i]);

**break**;

**case** "lista":

**if** (*isNumeric*(comandos[1]))

System.***out***.println(lista.longPalabras(Integer.*parseInt*(comandos[1])));

**else**

System.***err***.println("Comando no válido");

**break**;

**case** "borrar":

lista.clearAll();

**break**;

**case** "fin":

System.***out***.println("Fin del programa");

**break**;

**default**:

System.***err***.println("Comando no válido");

}

System.***out***.println("Resultado: " + lista);

} **while** (!(teclado.equals("fin")));

in.close();

}

**private** **static** **boolean** isNumeric(String cadena){

**try** {

Integer.*parseInt*(cadena);

**return** **true**;

} **catch** (NumberFormatException nfe){

**return** **false**;

}

}

}

## Solución profe con MAP

### Clase Ejercicio4

package unidad8.colecciones;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.NoSuchElementException;

import java.util.Scanner;

import java.util.Set;

public class Ejercicio4 {

static Palabras palabras = new Palabras();

public static void main(String[] args) throws IOException {

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

do {

System.out.print("> ");

} while (ejecutar(in.readLine()));

}

static boolean ejecutar(String linea) {

Scanner s = new Scanner(linea);

try {

String cmd = s.next("añadir:|lista|borrar$|borrar:|fin$");

switch (cmd) {

case "añadir:":

cmdAñadir(s);

break;

case "lista":

cmdLista(s);

break;

case "borrar":

case "borrar:":

cmdBorrar(s);

break;

case "fin":

return false;

}

} catch (NoSuchElementException e) {

System.out.println("no se reconoce el comando");

} finally {

s.close();

}

return true;

}

static void cmdAñadir(Scanner s) {

palabras.addPalabras(s.nextLine());

}

static void cmdBorrar(Scanner s) {

palabras.eliminarTodas();

if (s.hasNext())

cmdAñadir(s);

// palabras.addPalabras(s.nextLine());

}

static void cmdLista(Scanner s) {

try {

int longitud = s.nextInt();

if (s.hasNext())

System.out.println("error: no se esperaba: " + s.nextLine());

else {

Set<String> lista = palabras.get(longitud);

System.out.println(lista == null ? "no hay palabras de longitud " + longitud : lista);

}

} catch (NoSuchElementException e) {

System.out.println("error: no es un número entero: " + s.next());

};

}

}

### Clase Palabras

|  |
| --- |
| package unidad8.colecciones; |
|  | import java.util.Collections; |
|  | import java.util.Map; |
|  | import java.util.Scanner; |
|  | import java.util.Set; |
|  | import java.util.TreeMap; |
|  | import java.util.TreeSet; |
|  |  |
|  | public class Palabras { |
|  |  |
|  | private Map<Integer, Set<String>> mapa; |
|  |  |
|  | public Palabras() { |
|  | mapa = new TreeMap<>(); |
|  | } |
|  |  |
|  | public Palabras(String texto) { |
|  | this(); |
|  | addPalabras(texto); |
|  | } |
|  |  |
|  | private void add(String palabra) { |
|  | int longitud = palabra.length(); |
|  | Set<String> palabras = mapa.get(longitud); |
|  | if (palabras == null) |
|  | mapa.put(longitud, palabras = new TreeSet<>()); |
|  | palabras.add(palabra); |
|  | } |
|  |  |
|  | public void addPalabras(String texto) { |
|  | Scanner s = new Scanner(texto); |
|  | s.useDelimiter("\\P{L}+"); |
|  | while (s.hasNext()) |
|  | add(s.next()); |
|  | s.close(); |
|  | } |
|  |  |
|  | // public void addPalabras(String texto) { |
|  | // for(String palabra: texto.split("\\P{L}+")) |
|  | // add(palabra); |
|  | // } |
|  |  |
|  | public boolean contiene(String palabra) { |
|  | try { |
|  | return mapa.get(palabra.length()).contains(palabra); |
|  | } catch (NullPointerException e) { |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | // public String [] get(int longitud) { |
|  | // String [] a = new String[mapa.get(longitud).size()]; |
|  | // mapa.get(longitud).toArray(a); |
|  | // return a; |
|  | // } |
|  |  |
|  | public Set<String> get(int longitud) { |
|  | return Collections.unmodifiableSet(mapa.get(longitud)); |
|  | } |
|  |  |
|  | public void eliminarTodas() { |
|  | mapa.clear(); |
|  | } |
|  | } |

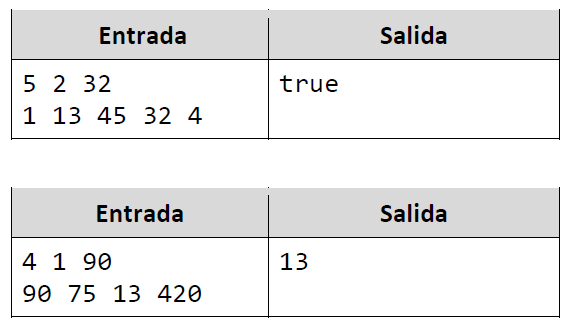
# Ejercicio 5

Programa para practicar las operaciones básicas con una cola. Tendrá que leer de la entrada estándar los datos siguientes:

* • Una línea en la que se escribirán 3 números enteros que llamaremos *N*, *K* y *X*.
* • Una línea en la que se escribirán *N* números, entre los que estará el número *X*.

Los *N* números leídos en la segunda línea se almacenarán en una cola y a continuación se retirarán *K* números. Finalmente se comprobará si el número *X* aún se encuentra almacenado en la cola. Si es así se mostrará *true* en la salida estándar. En caso contrario se mostrará el número más pequeño de los que permanecen en la cola. Si la cola estuviese vacía se mostrará el valor cero.

**Ejemplos:**



**package** unidad8\_colecciones;

**import** java.io.IOException;

**import** java.util.LinkedList;

**import** java.util.PriorityQueue;

**import** java.util.Queue;

**import** java.util.Scanner;

**public** **class** Ejercicio5 {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

System.***out***.println(">");

Queue<Integer> cola = **new** LinkedList<Integer>();

// Primera línea

Scanner in = **new** Scanner(System.***in***);

Scanner s = **new** Scanner(in.nextLine());

**int** n = s.nextInt();

**int** k = s.nextInt();

**int** x = s.nextInt();

System.***out***.println(">");

Scanner in2 = **new** Scanner(System.***in***);

**for** (**int** i=0; i<n; i++){

cola.add(in2.nextInt());

}

System.***out***.println(cola);

// Segunda línea

**for** (**int** i=0; i<k; i++){

cola.poll();

}

System.***out***.println(cola);

// Resultado

**if** (cola.isEmpty()) {

System.***out***.println("Cola vacía");

} **else** {

System.***out***.println("Tamaño cola: " + cola.size());

// Comprueba número

**if** (cola.contains(x)) {

System.***out***.println("Se encuentra (TRUE): " + x);

} **else** {

PriorityQueue<Integer> colaOrden = **new** PriorityQueue<Integer>(cola);

System.***out***.println("Número menor: " + colaOrden.peek());

}

}

System.***out***.println("FIN");

in.close();

in2.close();

}

}

# Ejercicio 6

Programa para manejar una lista de contactos telefónicos mediante una serie de comandos de texto, teniendo en cuenta que cada contacto puede tener varios teléfonos. El programa se ejecutará en la consola presentando una línea de comando que comenzará con el carácter > seguido de un espacio en blanco para indicarle al usuario que puede escribir uno de los comandos siguientes:

**Añadir un contacto:**

> nombre:*teléfono*

* • El dato *teléfono* estará formado únicamente por dígitos decimales.
* • No se pondrá límite al número de contactos que se pueden agregar ni a la cantidad de teléfonos que puede tener un contacto, salvo el que impone el tamaño de la memoria.
* • Si se especifica un nombre nuevo, se crea un nuevo contacto con el teléfono especificado.
* • Si el contacto ya existe, se le añade el teléfono especificado.
* • Si el teléfono ya existe para ese contacto, se mostrará el mensaje correspondiente.

**Buscar los teléfonos de un contacto:**

> buscar:nombre

En la línea siguiente se mostrarán los teléfonos del contacto separados por comas, o el mensaje correspondiente si éste no existe.

**Eliminar un contacto a partir de un nombre:**

> eliminar:nombre

Si el contacto existe, se pedirá confirmación para eliminarlo. Si el contacto no existe, se mostrará el mensaje correspondiente.

**Mostrar todos los contactos:**

> contactos

Se mostrarán todos los contactos, cada uno en una línea, en orden alfabético.

**Finalizar el programa:**

> salir

El programa mostrará los errores correspondientes si alguno de los comandos no se ajusta al formato especificado. En el caso de comandos del tipo *comando:datos*, se permiten espacios en blanco a derecha e izquierda de los dos puntos y al principio y final de la línea de comando.

Después de la ejecución de cada comando se volverá a mostrar la línea de comando.

**Ejemplo:**

> María:600000001

> Alberto:600000002

> Elena:600000003

> buscar:María

Teléfono: 600000001

> buscar:Fernando

El contacto no existe

> Fernando:600000004

> Elena:600000005

> buscar:Elena

600000003, 600000005

> salir